Available online at www.sciencedirect.com
ScienceDirect
Linear Algebra and its Applications 419 (2006) 107–124
LINEAR ALGEBRA AND ITS APPLICATIONS
www.elsevier.com/locate/laa

Principal minors, Part I: A method for computing all the principal minors of a matrix

Kent Griffin, Michael J. Tsatsomeros [*]

Mathematics Department, Washington State University, Pullman, WA 99164-3113, USA

Received 5 July 2005; accepted 14 April 2006
Available online 12 June 2006
Submitted by M. Neumann

Abstract

An order $O(2^n)$ algorithm for computing all the principal minors of an arbitrary $n \times n$ complex matrix is motivated and presented, offering an improvement by a factor of n^3 over direct computation. The algorithm uses recursive Schur complementation and submatrix extraction, storing the answer in a binary order. An implementation of the algorithm in MATLAB® is also given and practical considerations are discussed and treated accordingly.

AMS classification: 15A15; 65F40; 15A48; 68Q25; 90-08

Keywords: Principal submatrix; Schur complement; P-matrix; P-problem

1. Introduction

There are several instances and applications in the mathematical sciences where the principal minors of a matrix need be examined. Sometimes their exact value is needed and other times qualitative information, such as their signs, is required. Most notably, these instances include the detection of P-matrices (matrices with positive principal minors) as they appear in the study of the complementarity problem [1, Chapter 10], Cartan matrices of finite and affine type in Lie algebras [15], univalent differentiable mappings [16], as well as self-validating algorithms and

[*] Corresponding author.
E-mail addresses: kgriffin@math.wsu.edu (K. Griffin), tsat@math.wsu.edu (M.J. Tsatsomeros).

interval matrix analysis [2,14,17,18]. Other applications in which the values of the principal minors are of interest include the counting of spanning trees of a graph using the Laplacian, D-nilpotent automorphisms [10], as well as the solvability of the inverse multiplicative eigenvalue problem [8]. A related notoriously hard problem is the so-called *principal minor assignment problem* (see e.g., [11]), where a matrix with specified principal minors is sought (or its existence excluded). Solving this problem is the subject of the sequel to this paper, "Principal Minors, Part II: The principal minor assignment problem" [9].

The direct approach of evaluating all the principal minors of A via LU-factorizations entails a time complexity of $O(2^n n^3)$ [20]. As a result, the problems mentioned above share the tantalizing aspect of being exponentially hard. For instance, the detection of a P-matrix (known as the *P-problem*) is NP-hard [3,4]. The approach proposed in [20] regarding the P-problem offered an improvement to the tune of a factor of n^3, while at the same time being simple to implement and adaptable to computation in parallel. A similar "economization" in computing all the principal minors of a general matrix is proposed in this paper, resulting in the ability to study matrices of larger sizes even though the computation is inherently exponentially hard.

More specifically, in this paper we develop, implement and test an algorithm (MAT2PM) to compute all the principal minors of a given $n \times n$ complex matrix. MAT2PM is based on extending the reach and exploiting the computations of the algorithm in [20] (hereafter referred to as PTEST), which was designed to detect whether a given matrix is a P-matrix or not.

PTEST uses Schur complementation and submatrix extraction in a recursive manner to compute (up to) 2^n quantities. If in the course of PTEST any of these quantities is not positive, the algorithm terminates, declaring that the matrix at hand is not a P-matrix; otherwise it is a P-matrix. No further use of these 2^n quantities is made in PTEST, even when they all have to be computed; they are in fact overwritten. Moreover, in certain instances (e.g., in the presence at some stage of a P_0-matrix with zero trace) the original version of PTEST in [20] would not be able to proceed, for no Schur complement of a diagonal entry can be found.

In MAT2PM, several challenges of PTEST are resolved. First, employing the multilinearity of the determinant, the absence of a "pivot" (i.e., when all diagonal entries are zero) is overcome, giving us the ability to compute all the 2^n quantities that would be involved in a successful completion of PTEST. Second, these quantities are used to compute rationally all the principal minors of the initial matrix. Thirdly, MAT2PM is applicable to arbitrary complex matrices, including P-matrices and P_0-matrices. MAT2PM's output is then an array of all principal minors of the input matrix in a binary order. Lastly, care is taken for the robustness of MAT2PM as it relates to tolerance of zero pivots and zero principal minors, the minimization of round-off errors and ease of use.

An important aspect of the process underlying MAT2PM is its ability to be reversed and thus deal with the principal minor assignment problem. The resulting "PM2MAT" algorithm presented in the sequel referred to above builds directly on the MAT2PM algorithm of this paper.

In Section 2, we set forth the notation used in this paper. Section 3 presents the foundational work that led to the MAT2PM algorithm developed in Section 4. This is followed by two examples in Section 5 and conclusions in Section 6. The MATLAB®[1] codes found in the appendices are available for download on the web[2] or via email from the authors.

[1] Version 7.0.1.15 ($R14$) or later is required.

[2] http://www.math.wsu.edu/math/faculty/tsat/pm.html.

2. Notation and preliminaries

The following technical notation is used:

- $(A)_{ij}$ or A_{ij} is the (i, j)th entry of the matrix A. Similarly, $v_i = v(i)$ is the ith entry of the vector v.
- $\langle n\rangle = \{1, 2, \dots, n\}$ for every positive integer n.
- The lower case Greek letters α, β, γ are used as index sets. Thus, $\alpha, \beta, \gamma \subseteq \langle n\rangle$, and the elements of α, β, γ are assumed to be in ascending order. The number of elements in α is denoted $|\alpha|$.
- Let $\gamma \subseteq \langle n\rangle$ and $\beta = \{\beta_1, \beta_2, \dots, \beta_k\} \subseteq \langle|\gamma|\rangle$. Define the indexing operation $[\gamma]_\beta$ as
$$[\gamma]_\beta := \{\gamma_{\beta_1}, \gamma_{\beta_2}, \dots, \gamma_{\beta_k}\} \subseteq \gamma.$$
- $A[\alpha, \beta]$ is the submatrix of A whose rows and columns are indexed by $\alpha, \beta \subseteq \langle n\rangle$, respectively. When a row or column index set is empty, the corresponding submatrix is considered vacuous and by convention has determinant equal to 1.
- $A[\alpha] := A[\alpha, \alpha]$, $A(\alpha, \beta] := A[\alpha^c, \beta]$; $A[\alpha, \beta)$, $A(\alpha, \beta)$ and $A(\alpha)$ are defined analogously, where α^c is the complement of α with respect to the set $\langle n\rangle$. When $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$ is known explicitly (as in the examples), we let $A[\alpha_1, \alpha_2, \dots, \alpha_k] := A[\{\alpha_1, \alpha_2, \dots, \alpha_k\}]$.
- The *Schur complement* of an invertible principal submatrix $A[\alpha]$ in A is
$$A/A[\alpha] = A(\alpha) - A(\alpha, \alpha](A[\alpha])^{-1}A[\alpha, \alpha).$$

3. Detecting P-matrices and PTEST

For the purpose of developing and describing MAT2PM, we shall first discuss PTEST. Recall that an n-by-n complex matrix $A \in \mathscr{M}_n(\mathbb{C})$ is called a *P-matrix* (respectively, a *P_0-matrix*) if every principal minor of A is positive (respectively, nonnegative). We denote the class of P-matrices by $\mathbb{P}$ and the class of P_0-matrices by $\mathbb{P}_0$. For the general properties of these two matrix classes see e.g., [6, Chapter 5, pp. 131–134] or [13, Chapter 2, pp. 120–123]. We note that the P-matrices encompass such notable classes as the Hermitian positive definite matrices, the M-matrices, the totally positive matrices and the real diagonally dominant matrices with positive diagonal entries. The first systematic study of P-matrices appeared in the work of Fiedler and Ptak [7].

In [20] the following result is shown for real matrices; here we include the proof for completeness and in order to note that it is also valid for complex matrices.

Theorem 3.1. *Let $A \in \mathscr{M}_n(\mathbb{C})$ and $\alpha \subseteq \langle n\rangle$ with $|\alpha| = 1$. Then $A \in \mathbb{P}$ if and only if $A[\alpha]$, $A(\alpha)$, $A/A[\alpha] \in \mathbb{P}$.*

Proof. Without loss of generality, assume that $\alpha = \{1\}$. Otherwise we can consider a permutation similarity of A. If $A = [a_{ij}]$ is a P-matrix, then $A[\alpha]$ and $A(\alpha)$ are also P-matrices. That $A/A[\alpha]$ is a P-matrix is a well known fact (see e.g., [1, Exercise 10.6.1] or [19, Lemma 5.1]).

For the converse, assume that $A[\alpha] = [a_{11}]$, $A(\alpha)$ and $A/A[\alpha]$ are P-matrices. Using $a_{11} > 0$ as the pivot, we can row reduce A to obtain a matrix B with all of its off-diagonal entries in the first column equal to zero. As is well known, $B(\alpha) = A/A[\alpha]$. That is, B is a block triangular matrix whose diagonal blocks are P-matrices. It follows readily that B is a P-matrix. The determinant of any principal submatrix of A that includes entries from the first row of A coincides with the determinant of the corresponding submatrix of B and is thus positive. The determinant of any principal submatrix of A with no entries from the first row coincides with a principal minor of $A(\alpha)$ and is also positive. Hence A is a P-matrix. $\square$

The above theorem gives rise to the following algorithm for testing whether $A \in \mathscr{M}_n(\mathbb{C})$ is a P-matrix or not.

Algorithm 3.2. (PTEST)

```
    Function P($A$)
1.  Input $A = [a_{ij}] \in \mathscr{M}_n(\mathbb{C})$
2.  If $a_{11} \leq 0$ output '$A$ is not a P-matrix', stop
3.  Evaluate $A/A[1]$
4.  Call P($A(1)$)
    Call P($A/A[1]$)
5.  Output '$A$ is a P-matrix'
```

An essential part of MAT2PM consists of exploiting the values of the pivots a_{11} computed during the application of PTEST to an arbitrary matrix in order to compute its principal minors.

4. Finding all principal minors via MAT2PM

4.1. Preliminaries

It is well known that $\det(A) = \det(A[\alpha])\det(A/A[\alpha])$ [12].

The computations that MAT2PM performs require the following generalization of this result.

Lemma 4.1. *Let $A \in \mathscr{M}_n(\mathbb{C})$, $\alpha \subset \langle n\rangle$, where $A[\alpha]$ is nonsingular, and denote $\gamma = \alpha^c$. If $\beta \subseteq \langle|\alpha^c|\rangle$, then*

$$\det(A[\alpha \cup \beta']) = \det(A[\alpha])\det((A/A[\alpha])[\beta]),$$

where

$$\beta' = [\gamma]_\beta = \{\gamma_{\beta_1}, \gamma_{\beta_2}, \ldots, \gamma_{\beta_k}\}.$$

Proof. Without loss of generality, assume $\alpha = \{1, 2, \ldots, m\}$, $\beta = \{m+1, m+2, \ldots, m+k\}$; otherwise our considerations apply to a permutation similarity of A. Partition A into

$$A = \begin{bmatrix} B & C \\ D & E \end{bmatrix} = \begin{bmatrix} B_{11} & B_{12} & C_1 \\ B_{21} & B_{22} & C_2 \\ D_1 & D_2 & E \end{bmatrix}$$

with $B_{11} \in \mathscr{M}_m(\mathbb{C})$ and $B_{22} \in \mathscr{M}_k(\mathbb{C})$ $(m + k \leqslant n)$. Then the lemma amounts to noting that $B = A[\alpha \cup \beta']$ satisfies

$$\det(B) = \det(B_{11})\det((B/B_{11}))$$

which, in turn, implies

$$\det(B) = \det(B_{11})\det((A/B_{11})[1, 2, \ldots, k]). \qquad \square$$

To illustrate Lemma 4.1, suppose $A \in \mathscr{M}_6(\mathbb{C})$ has nonsingular principal submatrix $A[1, 3, 4]$. Then,

$$\begin{aligned}
\det(A[1, 2, 3, 4]) &= \det(A[1, 3, 4])\ (A/A[1, 3, 4])_{11},\\
\det(A[1, 3, 4, 5]) &= \det(A[1, 3, 4])\ (A/A[1, 3, 4])_{22},\\
\det(A[1, 2, 3, 4, 5]) &= \det(A[1, 3, 4])\ \det((A/A[1, 3, 4])[1, 2]).
\end{aligned}$$

In the MAT2PM algorithm to be described below, the set β is the singleton $\beta = \{j\}$ and so $\det((A/A[\alpha])[\beta]) = (A/A[\alpha])[j] = (A/A[\alpha])_{jj}$.

We also need a result about nested Schur complemention known as the *quotient property* of the Schur complement, which was first proved by Crabtree and Haynsworth [5].

Lemma 4.2. *Let $A \in \mathscr{M}_n(\mathbb{C})$, $\alpha \subset \langle n\rangle$. As in the previous lemma, let $\beta \subseteq \langle|\alpha^c|\rangle$ and $\beta' = [\alpha^c]_\beta$. If both $A[\alpha]$ and $A[\alpha \cup \beta']$ are nonsingular, then*

$$(A/A[\alpha \cup \beta']) = (A/A[\alpha])/((A/A[\alpha])[\beta]).$$

Once again, to illustrate Lemma 4.2, let $A \in \mathscr{M}_6(\mathbb{R})$ have all of its principal minors nonzero. Then,

$$\begin{aligned}
A/A[1, 2, 3, 4] &= (A/A[1, 3, 4])/((A/A[1, 3, 4])[1]),\\
A/A[1, 3, 4, 5] &= (A/A[1, 3, 4])/((A/A[1, 3, 4])[2]),\\
A/A[1, 2, 3, 4, 5] &= (A/A[1, 3, 4])/((A/A[1, 3, 4])[1, 2]).
\end{aligned}$$

If $A \in \mathscr{M}_n(\mathbb{C})$, then A has $2^n - 1$ principal minors. For computational simplicity and efficiency, these are recorded in a vector *pm* whose entries are ordered according to the following binary order.

Definition 4.3. Let $pm \in \mathbb{C}^{2^n-1}$ be a vector of the principal minors of $A \in \mathscr{M}_n(\mathbb{C})$. Further let i be an index of pm regarded as an n-bit binary number with

$$i = b_n b_{n-1} \dots b_3 b_2 b_1, \quad b_j \in \{0, 1\}, \quad j = 1, 2, \dots, n.$$

We say that the entries of pm are in *binary order* if

$$pm_i = \det(A[j_1, j_2, \dots, j_m]),$$

where $j_k \in \langle n\rangle$ are precisely those integers for which $b_{j_k} = 1$ for all $k = 1, 2, \dots, m$.

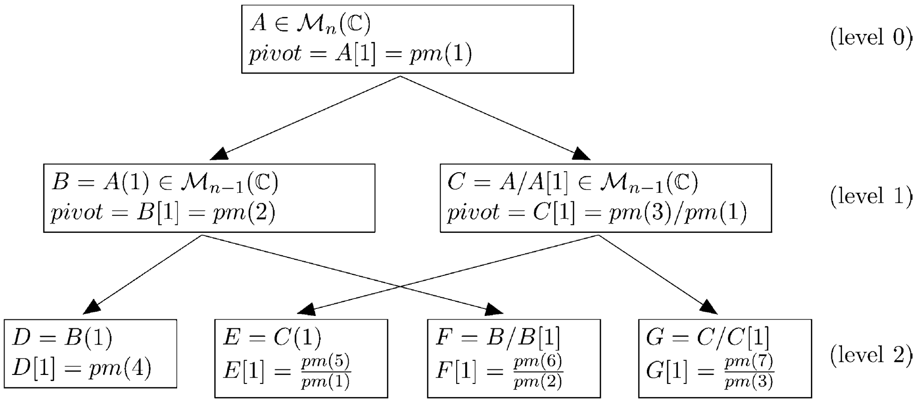
Remark 4.4. As a consequence of the definition of binary order, the entries of pm are as follows:

$$\begin{aligned}
pm = [&\det(A[1]), \det(A[2]), \det(A[1, 2]), \det(A[3]), \det(A[1, 3]),\\
&\det(A[2, 3]), \det(A[1, 2, 3]), \det(A[4]), \dots, \det(A)].
\end{aligned}$$

The program GETPM included in Appendix A can extract any desired principal minor from the vector pm of principal minors in binary order.

4.2. Description and analysis of MAT2PM

The algorithm implemented in MAT2PM is based on the recursive principle in PTEST and Proposition 4.6. It proceeds to find all the principal minors of an input matrix $A \in \mathscr{M}_n(\mathbb{C})$ in the binary order defined above by processing an input queue (called q in MAT2PM) of nq matrices of dimension $n1 \times n1$ and producing an output queue (called qq) of $2nq$ matrices of size $(n1-1) \times (n1-1)$. At each step in the algorithm, the $(1, 1)$ entry of each matrix in the input queue will either be a principal minor or will be a ratio of principal minors. Initially, the queue just contains the single $n \times n$ input matrix. We can schematically express the first 3 levels of the operation of the algorithm as follows:


$A \in \mathcal{M}_n(\mathbb{C})$
$pivot = A[1] = pm(1)$
(level 0)
$B = A(1) \in \mathcal{M}_{n-1}(\mathbb{C})$
$pivot = B[1] = pm(2)$
$C = A/A[1] \in \mathcal{M}_{n-1}(\mathbb{C})$
$pivot = C[1] = pm(3)/pm(1)$
(level 1)
$D = B(1)$
$D[1] = pm(4)$
$E = C(1)$
$E[1] = \frac{pm(5)}{pm(1)}$
$F = B/B[1]$
$F[1] = \frac{pm(6)}{pm(2)}$
$G = C/C[1]$
$G[1] = \frac{pm(7)}{pm(3)}$
(level 2)

The algorithm proceeds in *levels*. At $level = k$ we process 2^k matrices of size $(n-k) \times (n-k)$ to produce 2^k principal minors beginning with the 2^kth entry of pm.

Notice that in level 0, the $(1, 1)$ entry of the input queue matrix gives us all the principal minors of A involving rows and columns from $\{1\}$. In level 1, the $(1, 1)$ entries of the input queue matrices provide enough information to easily compute all the principal minors of A involving rows and columns from the index set $\{1, 2\}$, using the principal minor from level 0. In level 2, the $(1, 1)$ entries of the matrices of the current queue allow us to find all principal minors of A with indices from the set $\{1, 2, 3\}$, which involve the new index 3 using the principal minors produced in levels 0 and 1.

In general, if $level = k$, we can find all the principal minors of A, $\det(A[\alpha])$, with index sets of the form

$$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_{m-1}, k+1\},$$

where the index set of each principal minor we find contains the new index $k+1$ with all combinations of *smaller* indices, $\alpha_i < k+1$ for all $1 \leqslant i \leqslant m-1$. This is done by using the $(1, 1)$ entries of the matrices in the input queue combined with the principal minors found in all previous levels.

4.2.1. Pivots

Each $(1, 1)$ entry of a matrix in a processing queue is referred to as a *pivot* in the code and in the description to follow. A pivot is a principal minor, if it comes from the first matrix in a queue on a given level. Otherwise, a pivot is the ratio of two principal minors as explained in the next subsection. Schur complements are indeed taken with respect to the pivots.

4.2.2. Theoretical basis for MAT2PM

Producing the output queue from the input queue of matrices requires repeated application of Lemmas 4.1 and 4.2. For example, let us consider producing the matrices $\{D, E, F, G\}$ from the matrices $\{B, C\}$ in the schematic above when computing level 2 from level 1. The first matrix is just a submatrix of a submatrix, so $D_{11} = A_{33}$ is just the 1×1 principal minor corresponding to the next diagonal entry of A. For all the other matrices, we need to apply Lemma 4.1. For example, $E = (A/A[1])(1)$ so

$$E[1] = E_{11} = (A/A[1])_{22} = \det(A[1,3])/\det(A[1]) = pm(5)/pm(1).$$

Similarly, $F = A(1)/A[2]$ and applying Lemma 4.1,

$$F_{11} = (A/A[2])_{22} = \det(A[2,3])/\det(A[2]) = pm(6)/pm(2).$$

To produce matrix G from matrix C, however, we first need to apply Lemma 4.2, obtaining $G = (A/A[1])/((A/A[1])[1]) = A/A[1,2]$. Then, applying Lemma 4.1, $G_{11} = (A/A[1,2])_{11} = \det([A[1,2,3])/\det(A[1,2])$. Each time we take Schur complements of Schur complements in the algorithm, we must first apply Lemma 4.2 before using Lemma 4.1 to obtain a ratio of principal minors.

Remark 4.5. Note that the left half of the output queue is obtained by deleting the first row and column of each matrix in the input queue and putting the resulting matrix in the output queue in the same order. The right half of the output queue is computed by taking the Schur complement of each matrix in the input queue with respect to its $(1,1)$ entry and then placing the result in the output queue in the same order.

The following result provides the theoretical basis for the functionality of MAT2PM.

Proposition 4.6. *Let $A \in \mathscr{M}_n(\mathbb{C})$. Consider the pivots produced by the algorithm described above ordered from level* 0 *to level $n-1$, then from left to right. The numerators of the pivots are the principal minors of A in binary order. The denominators of the pivots at each level equal* 1 (*for the first pivot of each level*), *followed by the principal minors of all previous levels in binary order*; *see* (4.1).

Proof. We argue by induction on the $level$. When $level = 0$, the single principal minor $pm_1 = A[1]$ is in binary order. Since this is the first (and only) pivot, its numerator is 1.

Assume the pivots in $level = k$ have the described form. As observed previously, note that the principal minors in the numerators of the pivots all involve the index $k+1$. Thus, in order, level k has 2^k pivots,

$$\left\{\det(A[k+1]), \frac{\det(A[1,k+1])}{\det(A[1])}, \frac{\det(A[2,k+1])}{\det(A[2])}, \ldots, \frac{\det(A[1,2,\ldots,k,k+1])}{\det(A[1,2,\ldots,k])}\right\}. \tag{4.1}$$

We now form $level = k+1$ with 2^{k+1} pivots. The first 2^k of these are formed by taking the submatrices formed by removing the first row and column of each matrix at $level = k$. Thus, by Lemma 4.1, the first (left most) 2^k pivots are:

$$\left\{\det(A[k+2]), \frac{\det(A[1,k+2])}{\det(A[1])}, \frac{\det(A[2,k+2])}{\det(A[2])}, \ldots, \frac{\det(A[1,2,\ldots,k,k+2])}{\det(A[1,2,\ldots,k])}\right\}. \tag{4.2}$$

Since Lemma 4.1 gives

$$(A/A[j_1, j_2, \ldots, j_m])[i] = \frac{\det(A[j_1, j_2, \ldots, j_m, j_m + 1])}{\det(A[j_1, j_2, \ldots, j_m])},$$

where i corresponds with $j_m + 1$, it follows that

$$(A/A[j_1, j_2, \ldots, j_m])[i + 1] = \frac{\det(A[j_1, j_2, \ldots, j_m, j_m + 2])}{\det(A[j_1, j_2, \ldots, j_m])}.$$

Therefore, the first 2^k pivots of $level = k + 1$ have the prescribed binary order for their numerators and the denominators remain in the order they were in at $level = k$.

To produce the right most 2^k matrices of $level = k + 1$, we take Schur complements of each of the matrices from $level = k$. By Lemma 4.2, this has the effect of adding $k + 1$ to the index set α, then computing $A/A[\alpha]$ for each matrix in the input queue. Then, if we take the pivots of these new matrices, we see by Lemma 4.1 that the numerators of the pivots are the principal minors we get by *appending* $k + 2$ to the index set of the principal minors. Also, by taking the Schur complement, the principal minors of the denominators have $k + 1$ appended to them. Thus, as before, if the pivots of level k are

$$\left\{\det(A[k + 1]), \frac{\det(A[1, k + 1])}{\det(A[1])}, \frac{\det(A[2, k + 1])}{\det(A[2])}, \ldots, \frac{\det(A[1, 2, \ldots, k, k + 1])}{\det(A[1, 2, \ldots, k])}\right\},$$

after taking Schur complements the pivots formed by taking the $(1, 1)$ entries of the new matrices have the form

$$\left\{\frac{\det(A[k + 1, k + 2])}{\det(A[k + 1])}, \frac{\det(A[1, k + 1, k + 2])}{\det(A[1, k + 1])}, \frac{\det(A[2, k + 1, k + 2])}{\det(A[2, k + 1])}, \ldots\ldots, \frac{\det(A[1, 2, \ldots, k + 1, k + 2])}{\det(A[1, 2, \ldots, k, k + 1])}\right\}. \tag{4.3}$$

Concatenating Eq. (4.2) with (4.3), we see that level $k + 1$ also has the desired order for both the numerators and denominators of the pivots. □

5. Examples

Example 5.1. Suppose by way of example that we wish to use MAT2PM to find the principal minors of

$$A = \begin{bmatrix} 1 & 2 & 6 \\ 2 & 4 & 5 \\ -1 & 2 & 3 \end{bmatrix}.$$

Since $A \in \mathscr{M}_3(\mathbb{R})$, we expect the output of MAT2PM to be a vector in $\mathbb{R}$ with $2^3 - 1 = 7$ entries, having the form

$$pm = [\det(A[1]), \det(A[2]), \det(A[1, 2]), \det(A[3]), \det(A[1, 3]), \det(A[2, 3]), \det(A)].$$

For simplicity, set the matrix pseudo-pivot variable *ppivot* to 1. The value of this variable will be used instead of a pivot value if the pivot is zero or near zero, since one cannot take Schur complements with respect to a matrix whose determinant is zero. Then, initially when $level = 0$ the input queue of matrices q just has 1 matrix:

$$q_1 = \begin{bmatrix} 1 & 2 & 6 \\ 2 & 4 & 5 \\ -1 & 2 & 3 \end{bmatrix}.$$

Level 0

MAT2PM processes this matrix q_1. First, $pm_1 = (q_1)_{11}$, and we have found the first principal minor $\det(A[1])$. Then, following the outline above, create a new queue of smaller matrices qq by taking a submatrix $qq_1 = q_1(1)$ and the Schur complement $qq_2 = q_1/q_1[1]$. Next let $q = qq$ and at the end of the first main level loop we have

$$q_1 = \begin{bmatrix} 4 & 5 \\ 2 & 3 \end{bmatrix}, \quad q_2 = \begin{bmatrix} 0 & -7 \\ 4 & 9 \end{bmatrix},$$
$$pm = [1].$$

Level 1

Next, $pm_2 = (q_1)_{11}$ which equals $\det(A[2])$. Then, compute $qq_1 = q_1(1)$ and $qq_3 = q_1/q_1[1]$, where, in practice, we produce both matrices in the output queue that derive from a given input queue matrix at the same time for efficiency. Recall that $q_1/q_1[1]$ is stored in qq_3 and not qq_2 to preserve the binary ordering of the principal minors we compute from the (1, 1) entries of the output queue matrices.

Next, we process q_2. Now, $pm_3 = ((q_2)_{11} + ppivot)pm_1$, where $ppivot = 1$ for this example. In this computation, two new factors come into play.

Computing principal minors from pivots

Since the pivots produced by taking Schur complements are not the principal minors directly, but are ratios of principal minors, we need to multiply the pivot of the matrix by pm_1 applying Lemma 4.1. Note that due to the structure of the algorithm, each previously produced principal minor will be used as a factor in producing the next level of principal minors exactly once, in the order they occur in pm.

Handling zero pivots

Second, since $(q_2)_{11} = 0$, we add $ppivot$ to pm_3, which makes it possible to take the next Schur complement. Therefore,

$$q_2 = \begin{bmatrix} 1 & -7 \\ 4 & 9 \end{bmatrix}$$

for purposes of taking the Schur complement of q_2. We add 3, the index of the principal minor entry that was changed from zero to $ppivot$, to a vector called $zeropivs$ so that we can perform the additional operations necessary to produce the actual principal minors of the matrix A.

Now, we can take the submatrix $qq_2 = q_2(1)$ and Schur complement $qq_4 = q_2/q_2[1]$. Then, letting $q = qq$, at the end of the second outer loop we have:

$$q_1 = [3], \quad q_2 = [9], \quad q_3 = [1/2], \quad q_4 = [37],$$
$$pm = [1, 4, 1].$$

Level 2

In the final iteration of the main level loop we do not compute any further submatrices since the input matrices are already 1×1. First, we set $pm_4 = (q_1)_{11}$. All the remaining principal minors are computed by multiplying the remaining pivots by the previously computed principal minors, in the order they were computed. Thus,

$$pm_5 = (q_2)_{11} \cdot pm_1, \quad pm_6 = (q_3)_{11} \cdot pm_2 \quad \text{and} \quad pm_7 = (q_4)_{11} \cdot pm_3.$$

We exit the main loop with

$$pm = [1, 4, 1, 3, 9, 2, 37].$$

Zero pivot loop

Finally, we enter the zero pivot loop with one entry in $zeropivs$, a 3. The details of this loop are complex, but the concept is simple. Due to the multilinearity of determinants with respect to a given row, we can correct any places we added $ppivot$ to a pivot by subtracting a principal minor we have computed from any descendants of the zero pivot. In this case, we compute $pm_3 = 0$ by undoing the effects of adding $ppivot$ to this principal minor. The algebra for doing this in MAT2PM provides additional accuracy in cases where the pivot is not exactly zero. Then, we subtract $pm_5 = \det(A[1, 3])$ from $pm_7 = \det(A[1, 2, 3])$, which undoes the effects of using a false pivot for $pm_3 = \det(A[1, 2])$. This follows since if $B = A/A[1]$,

$$\det \begin{bmatrix} 0 & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \det \begin{bmatrix} 1 & b_{12} \\ b_{21} & b_{22} \end{bmatrix} - \det \begin{bmatrix} 1 & 0 \\ b_{21} & b_{22} \end{bmatrix} \Rightarrow pm_7 = \widehat{pm_7} - pm_5,$$

where we use $\widehat{pm_7}$ to represent the false intermediate value of pm_7 resulting from using a false value of 1 instead of 0 for pm_3.

Only principal minors involving both entries of the set $\alpha = \{1, 2\}$ that descend from the Schur complement of q_2 of the second level need this type of correction. So,

$$pm = [1, 4, 0, 3, 9, 2, 28]$$

is the final vector of correct principal minors of A.

Example 5.2. To demonstrate the handling of zero pivots in more detail, we now consider using MAT2PM to find all the principal minors of

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

Again, for simplicity the pseudo-pivot variable $ppivot$ is assumed to be 1. We use the notation $0 \to 1$ to indicate when we add $ppivot$ to make a pivot nonzero.

Level 0

Given input queue

$$q_1 = \begin{bmatrix} 0 \to 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

we produce the principal minor array of

$$pm = [1],$$

where $zeropivs = [1]$ contains the index the entry in pm that we have changed from 0 to $ppivot = 1$.

Then we can compute the submatrix $A(1)$ and the Schur complement $A/A[1]$ to produce output queue

$$qq_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad qq_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}.$$

Level 1

After letting $q = qq$ we begin the next level loop with the pivots of both input matrices equal to 0. Adding 1 to these we produce:

$$q_1 = \begin{bmatrix} 0 \to 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad q_2 = \begin{bmatrix} 0 \to 1 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix},$$

$$pm = [1, 1, 1],$$

$$zeropivs = [1, 2, 3],$$

$$qq_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad qq_2 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad qq_3 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad qq_4 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Level 2

At the beginning of the level loop again each pivot is 0, so we compute:

$$q_1 = \begin{bmatrix} 0 \to 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad q_2 = \begin{bmatrix} 0 \to 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad q_3 = \begin{bmatrix} 0 \to 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad q_4 = \begin{bmatrix} 0 \to 1 & 1 \\ 1 & 0 \end{bmatrix},$$

$$pm = [1, 1, 1, 1, 1, 1, 1],$$

$$zeropivs = [1, 2, 3, 4, 5, 6, 7],$$

$$qq_1 = [0], \quad qq_2 = [0], \quad qq_3 = [0], \quad qq_4 = [0],$$

$$qq_5 = [0], \quad qq_6 = [0], \quad qq_7 = [0], \quad qq_8 = [-1].$$

Level 3

Since the input queue consists entirely of 1×1 matrices all that remains is to append the entries of the output queue matrices to the principal minor vector to obtain

$$pm = [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, -1].$$

Zero pivot loop

With the principal minor vector pm computed above, along with $ppivot = 1$ and $zeropivs = [1, 2, 3, 4, 5, 6, 7]$, we have the necessary information to perform the computations that will make pm correct for the given matrix. To prevent division by 0 we "undo" the effects of adding $ppivot$ to our zero pivots in the opposite order that we applied them. Reading the $zeropivs$ vector of indices in reverse order, the first time through the loop we set $mask = zeropivs(7) = 7$. When we computed $pm(7)$ in level 2, $pivot = pm(7)/pm(3)$ was zero, so we added $ppivot = 1$ to $pivot$. Using $\widehat{pm(7)}$ to indicate modified or false principal minor values, then

$$\widehat{pm(7)} = (pm(7)/pm(3) + ppivot)pm(3)$$

since we add the $ppivot$ to the $pivot$ when we take the Schur complement which occurs before converting pivots to principal minors by multiplying by previous principal minors. Therefore it follows that

$$pm(7) = (\widehat{pm(7)}/pm(3) - ppivot)pm(3).$$

In our example, $pm(3) = 1$ from previously adding $ppivot$ to $pm(3)/pm(1)$ and $ppivot = 1$ so we effectively set $pm(7) = 0$.

Similarly we set $pm(6) = 0$ when $mask = zeropivs(6) = 6$ and continuing this process we obtain the final vector of principal minors

$$pm = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1].$$

Any time we change a *pivot* of a matrix we change the principal minors that are computed from any Schur complement of that matrix. However, the multilinearity of the determinant always allows us to correct for this by taking the appropriate difference of principal minors. Thus in our example where $A = [a_{ij}]$ and $a_{11} = 0$, since

$$\det\begin{bmatrix} 0 & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{24} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \det\begin{bmatrix} ppivot & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} - \det\begin{bmatrix} ppivot & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

we must correct for using *ppivot* instead of 0 for a_{11} by letting $pm(15) = \widehat{pm(15)} - ppivot \cdot pm(14)$, recalling that $pm(15) = \det(A[1, 2, 3, 4])$ and $pm(14) = \det([A[2, 3])$. This is done in an embedded loop at the bottom of the zero pivot loop. In this example, all the principal minors being subtracted are zero, so no change results to the principal minor array.

6. Complexity and conclusions

6.1. Time complexity and practical issues

It has been shown that the time complexity of the PTEST algorithm is $O(2^n)$ [20, Theorem 3.3]. For the generic case in which there are no zero pivots, MAT2PM only adds (slightly less than) one multiply per principal minor produced to the complexity of the PTEST algorithm. This multiply converts the pivots (those that are not diagonal entries of the input matrix A) into principal minors. Therefore, MAT2PM has time complexity $O(2^n)$ also. This is a considerable improvement over the $O(2^n n^3)$ complexity that results from naively computing each of the $2^n - 1$ determinants of $A \in \mathscr{M}_n(\mathbb{C})$ independently [20, p. 411].

MAT2PM also has an $O(2^n)$ memory requirement just in order to store the output.

The practical results of this are that on a fairly typical computer (in 2005: MATLAB® R14 on Windows XP, 2.6 GHz Pentium 4 Processor, 512 MB memory) one may find the approximately 1 million principal minors of a random 20×20 real matrix in about 15 s using MAT2PM. If one naively computes the same million principal minors by calling Matlab's *det* function independently for each minor, the same computation will take about 1380 s.

The fundamental computational activity of MAT2PM is taking Schur complements with respect to a single entry of a matrix. This involves an outer product and matrix difference for half of the output matrices of the output queue. Therefore, the individual computations MAT2PM performs are quite light, but there are many of them, and there is considerable overhead in the current straightforward implementation from data movement. We believe that the performance of MAT2PM would benefit greatly from careful implementation in a lower level language calling an optimized basic linear algebra library.

The MAT2PM algorithm is able to speed up the computation of all principal minors of a matrix by reusing a given Schur complement to obtain all the principal minors that Lemma 4.1 implies while using Lemma 4.2 to speed the computation of Schur complements with larger index sets. The price for doing this is that MAT2PM cannot do traditional partial pivoting. However, MAT2PM can avoid using extremely small pivots by setting a threshold (*thresh*) below which MAT2PM resorts to the slower but more accurate pseudo-pivot code.

By default this threshold is set to 10^{-5} times the average magnitude of the values in the matrix. This is fairly conservative and has been found to provide usable accuracy in many situations (for example, the maximum relative error for all the principal minors of a random, real 14×14 with entries chosen from $(0, 1)$ is typically less than $2.0\text{E}{-10}$ and pseudo pivoting does not occur with default settings). Note that setting *thresh* to extremely large values will negatively impact performance, while setting it to extremely small values could result in numerical inaccuracies.

For the convenience of the user, MAT2PM outputs the number of times pseudo-pivoting was employed and the magnitude of the smallest Schur complement pivot used. Also note that principal minors near zero are subject to larger relative error and can be verified using an explicit call to *det*.

Since the complexity of MAT2PM is of $O(2^n)$, we find, using the same computer on which we can compute the principal minors of a 20×20 real matrix in 15 s, that we can compute all the principal minors of a 21×21 real matrix in 30 s. Similarly, we can compute all the principal minors of a 22×22 real matrix in about 1 minute. However, for larger matrices the memory available to MAT2PM is exhausted, paging to disk occurs and performance suffers dramatically. Thus, finding all the principal minors of a 24×24 matrix takes 443 s which is much longer than $15 \times 2^4 = 240$ s which we would expect if the time complexity only grew at $O(2^n)$.

6.2. Conclusions

In this paper a structured method to present and compute all the principal minors of a real or complex matrix is presented, which reuses much of the work done to compute "smaller" principal minors (those that require shorter index sets to describe them) to produce "larger" ones, the determinant of the matrix being the last principal minor that the algorithm computes. This reduces the time complexity to compute these minors from $O(2^n n^3)$ to $O(2^n)$. For large matrices (say larger than 10×10) this represents a considerable time savings over computing the minors independently. Zero or nearly zero principal minors are handled at a performance penalty.

6.3. Future work

The algorithm of MAT2PM not only implements an efficient algorithm for computing all the principal minors of a matrix, but it also provides a structure for solving the principal minor assignment problem. This work is the content of the sequel to this paper [9].

Appendix A. MAT2PM

```
% MAT2PM Finds principal minors of an n x n matrix.
% PM = MAT2PM(A)
% where "A" is an n x n matrix in which zero can arise as a pivot at any
% point.  MAT2PM returns a 2^n - 1 vector of all the principal minors
% of the matrix "A".
%
% PM = MAT2PM(A, THRESH)
% Explicitly sets the pseudo-pivot threshold to THRESH.  Pseudo-pivoting
% will occur when a pivot smaller in magnitude than THRESH arises.  Set
% THRESH = 0 to never pseudo-pivot except for a pivot of exactly zero.
%
% The structure of PM, where |A[v]| is the principal minor of "A" indexed
% by the vector v:
% PM: |A[1]|, |A[2]|, |A[1 2]|, |A[3]|, |A[1 3]|, |A[2 3]|, |A[1 2 3]|,...

function [pm] = mat2pm(a, thresh)
% Only works on up to 48x48 matrices due to restrictions
% on bitcmp and indices.

n = length(a);
scale = sum(sum(abs(a)))/(n*n); % average magnitude of matrix
if scale == 0
    scale = 1;          % prevent divide by 0 if matrix is zero
end
ppivot = scale;         % value to use as a pivot if near 0 pivot arises
if nargin == 1
    thresh = (1.0e-5)*scale;    % when to pseudo-pivot
end



zeropivs = [];
pm = zeros(1, 2^n - 1);    % where the principal minors are stored
ipm = 1;                   % index for storing principal minors
q = zeros(n,n,1);          % q is the input queue of unprocessed matrices
q(:,:,1) = a;              % initial queue just has 1 matrix to process
pivmin = inf;              % keep track of smallest pivot actually used

%
% Main 'level' loop
```

```
%
for level = 0:n-1
    [n1, n1, nq] = size(q);
    % The output queue has twice the number of matrices, each one smaller
    % in row and col dimension
    qq = zeros(n1-1, n1-1, nq*2);
    ipm1 = 1;                  % for indexing previous pm elements
    for i = 1:nq
        a = q(:,:,i);
        pm(ipm) = a(1,1);
        if n1 > 1
            abspiv = abs(pm(ipm));
            if abspiv <= thresh
                zeropivs = union(zeropivs, ipm);
                % Pivot nearly zero, use "pseudo-pivot"
                pm(ipm) = pm(ipm) + ppivot;
                abspiv = abs(pm(ipm));
            end
            if abspiv < pivmin
                pivmin = abspiv;
            end
            b = a(2:n1,2:n1);
            d = a(2:n1,1)/pm(ipm);
            c = b - d*a(1,2:n1);

            % Order the output queue to make the elements of pm come out
            % in the correct order.
            qq(:,:,i) = b;
            qq(:,:,i+nq) = c;
        end
        if i > 1
            % if i > 1, to convert from a general pivot to a principal
            % minor, we need to multiply by every element of the pm matrix
            % we have already generated, in the order that we generated it.
            pm(ipm) = pm(ipm)*pm(ipm1);
            ipm1 = ipm1 + 1;
        end
        ipm = ipm + 1;
    end
    q = qq;
end


%
% Zero Pivot Loop
%
% Now correct principal minors for all places we used ppivot as a pivot
% in place of a (near) 0.
for i = length(zeropivs):-1:1
    mask = zeropivs(i);
    delta = msb(mask);
    delta2 = 2*delta;
    ipm1 = bitand(mask, bitcmp(delta,48));
    if ipm1 == 0
        pm(mask) = pm(mask) - ppivot;
    else
        pm(mask) = (pm(mask)/pm(ipm1) - ppivot)*pm(ipm1);
    end
    for j = mask+delta2:delta2:2^n - 1
        pm(j) = pm(j) - ppivot*pm(j - delta);
    end
end
% Warn user in case larger pivots are desired
fprintf(2, 'MAT2PM: pseudo-pivoted %d times, smallest pivot used:
%e\n', ...
    length(zeropivs), pivmin);

% Returns the numerical value of the most significant bit of x.
% For example, msb(7) = 4, msb(6) = 4, msb(13) = 8.
function [m] = msb(x)
persistent MSBTABLE     % MSBTABLE persists between calls to mat2pm
if isempty(MSBTABLE)
    % If table is empty, initialize it
    MSBTABLE = zeros(255,1);
    for i=1:255
        MSBTABLE(i) = msbslow(i);
    end
end

m = 0;
% process 8 bits at a time for speed
if x ~= 0
    while x ~= 0
        x1 = x;
        x = bitshift(x, -8);    % 8 bit left shift
        m = m + 8;
    end
    m = bitshift(MSBTABLE(x1), m-8); % right shift
end


% Returns the numerical value of the most significant bit of x.
% For example, msb(7) = 4, msb(6) = 4, msb(13) = 8.  Slow version
% used to build a table.
```

```
function [m] = msbslow(x)
m = 0;
if x ~= 0
    m = 1;
    while  x ~= 0
        x = bitshift(x, -1);
        m = 2*m;
    end
    m = m/2;
end
```

Appendix B. GETPM

```
% GETPM  Extracts a desired principal minor from the PM structure produced
% by MAT2PM.
%   PMINOR = GETPM(PM, V)
%   where PM is a vector of 2^n - 1 principal minors in binary order
%   produced by MAT2PM and V is the vector of the index set.  The elements
%   of V need not be sorted, but they must all be unique.
%
%   Example: If
%   A = rand(6);
%   pm = mat2pm(A);
%
%   then
%
%   getpm(pm, [1, 3, 5])
%
%   produces the same result as
%
%   det(A([1, 3, 5],[1, 3, 5]))
function [pminor] = getpm(pm, v)
% The index into pm is simply the binary number with the v(i)'th bit set
for each i.
n = length(v);        % length of vector containing indices of minor
idx = 0;
for i = 1:n
    idx = idx + bitshift(1,v(i)-1);
end
pminor = pm(idx);
```

References

[1] A. Berman, R.J. Plemmons, Nonnegative Matrices in the Mathematical Sciences, SIAM, Philadelphia, 1994.
[2] X. Chen, Y. Shogenji, M. Yamasaki, Verification for existence of solutions of linear complementarity problems, Linear Algebra Appl. 324 (2001) 15–26.

[3] G.E. Coxson, The P-matrix problem is co-NP-complete, Math. Programm. 64 (1994) 173–178.
[4] G.E. Coxson, Computing exact bounds on elements of an inverse interval matrix is NP-hard, Reliable Comput. 5 (1999) 137–142.
[5] D.E. Crabtree, E.V. Haynsworth, An identity for the Schur complement of a matrix, Proc. Amer. Math. Soc. 22 (1969) 364–366.
[6] M. Fiedler, Special Matrices and Their Applications in Numerical Mathematics, Martinus Nijhoff Publishers, Dordrecht, 1986.
[7] M. Fiedler, V. Pták, Some generalizations of positive definiteness and monotonicity, Numer. Math. 9 (1966) 163–172.
[8] S. Friedland, On inverse multiplicative eigenvalue problems for matrices, Linear Algebra Appl. 12 (1975) 127–137.
[9] K. Griffin, M.J. Tsatsomeros, Principal minors, Part II: The principal minor assignment problem, Linear Algebra Appl., in press, doi:10.1016/j.laa.2006.04.009.
[10] G. Gorni, T.-G. Halszka, G. Zampieri, Druzkowski matrix search and D-nilpotent automorphisms, Indag. Math. (N.S.) 10 (1999) 235–245.
[11] O. Holtz, H. Schneider, Open problems on GKK τ-matrices, Linear Algebra Appl. 345 (2002) 263–267.
[12] R.A. Horn, C.R. Johnson, Matrix Analysis, Cambridge University Press, New York, 1985.
[13] R.A. Horn, C.R. Johnson, Topics in Matrix Analysis, Cambridge University Press, New York, 1991.
[14] C. Jansson, J. Rohn, An algorithm for checking regularity of interval matrices, SIAM J. Matrix Anal. Appl. 20 (1999) 756–776.
[15] V.G. Kac, Infinite Dimensional Lie Algebras, Cambridge University Press, New York, 1990.
[16] C. Olech, T. Parthasarathy, G. Ravindran, A class of globally univalent differentiable mappings, Arch. Math. (Brno) 26 (1990) 165–172.
[17] J. Rohn, G. Rex, Interval P-matrices, SIAM J. Matrix Anal. Appl. 17 (1996) 1020–1024.
[18] S. Rump, Self-validating methods, Linear Algebra Appl. 324 (2001) 3–13.
[19] M. Tsatsomeros, Principal pivot transforms: properties and applications, Linear Algebra Appl. 300 (2000) 151–165.
[20] M. Tsatsomeros, L. Li, A recursive test for P-matrices, BIT 40 (2000) 404–408.